ObjectStore Release Notes

Release 6.1

February 2003

ObjectStore Release Notes

ObjectStore Release 6.1 for all platforms, February 2003

Contents

Preface

ObjectStore® is an object-oriented database management system suited for rapid application development and deployment in multitiered environments. It combines the data query and management capabilities of a traditional database with the flexibility and power of C++ and Java interfaces.

Purpose

This document describes changes to ObjectStore for Release 6.1.

Audience

This document is for administrators or developers responsible for the installation and maintenance of ObjectStore. It is assumed that you are familiar with the ObjectStore host platform and comfortable using the operating system.

Installing this release

For information about installing Release 6.1, see one of the following:

- *ObjectStore Installation for Windows*
- *ObjectStore Installation for UNIX*

Notation Conventions

This document uses the following conventions:

Convention	*Meaning*
`Courier`	`Courier font` indicates code, syntax, file names, API names, system output, and the like.
`Bold Courier`	**`Bold Courier font`** is used to emphasize particular code.
`Italic Courier`	*`Italic Courier font`* indicates the name of an argument or variable for which you must supply a value.
Sans serif	Sans serif typeface indicates the names of user interface elements such as dialog boxes, buttons, and fields.
Italic serif	In text, *italic serif typeface* indicates the first use of an important term.
[ ]	Brackets enclose optional arguments.
{ *a* | *b* | *c* }	Braces enclose two or more items. You can specify only one of the enclosed items. Vertical bars represent OR separators. For example, you can specify *a* or *b* or *c*.
. . .	Three consecutive periods indicate that you can repeat the immediately previous item. In examples, they also indicate omissions.

Obtaining Support

To obtain information about purchasing technical support, contact your local sales office listed at www.objectstore.net/contact (worldwide) or call 1-800-962-9620 (in the United States).

Technical Support

When you purchase technical support, the following services are available to you:

- You can send questions to support@objectstore.net. Remember to include your site ID in the body of the email message.
- You can call the Technical Support organization to get help resolving problems. If you are in North America, call 781-280-4005. If you are outside North America, refer to the Technical Support Web site.

- You can file a report or question with Technical Support by going to www.objectstore.net/support.
- You can access the Technical Support Web site, which includes
  - A template for submitting a support request. This helps you provide the necessary details, speeding up response time.
  - Frequently asked questions (FAQs) that you can browse and query.
  - Online documentation for all ObjectStore products.
  - White papers and short articles about using ObjectStore products.
  - Sample code and examples.
  - The latest versions of ObjectStore products, service packs, and publicly available patches for downloading.
  - Access to a support matrix that lists platform configurations supported by this release; see `www.objectstore.net/support/matrix`.
  - Support policies.
  - Local phone numbers and hours when support personnel can be reached.

Education Services

Use the ObjectStore education services site (`www.objectstore.net/services/education`) to learn about the standard course offerings and custom workshops.

If you are in North America, you can call 1-800-477-6473 x4452 to register for classes; if you are outside North America, refer to the Technical Support Web site. For information on current course offerings or pricing, send e-mail to `classes@progress.com`.

Web-Accessible Documentation

The www.objectstore.net/documentation Web site provides access to a full set of product documentation for the current and one previous supported release. To navigate to the documentation page, click Support for the support page, and then click Documentation. A search utility enables you to search the documents for specific information. READMEs for every Service Pack release to the present time are also available from this location. On occasion, you are likely to find additional information or documentation clarification posted between releases.

Your Comments

ObjectStore product development welcomes your comments about its documentation. Send any product feedback to support@objectstore.net. To expedite your documentation feedback, begin the subject with `Doc:`. For example:

```
Subject: Doc: Incorrect message on page 76 of reference manual
```

Release Information

This release document describes Release 6.1 of ObjectStore. It includes the following sections:

New and Changed Features

The following sections describes features of ObjectStore that were added or changed for Release 6.1. The last section, Summary of Changes on page 12, summarizes all user-visible for this release.

ossg's Default Front-End Parser

Previous to Release 6.1, the ObjectStore schema generation utility (`ossg`) required you to use the `-edgfe` option to enable an improved front-end parser that provides improved language support, including better support for nested classes and templates. Starting with Release 6.1, this parser is the default, and the `-edgfe` option is no longer recognized. If you wish to use the pre-Release 6.1 front-end parser, you must invoke `ossg` with the `-auditor` option. Note, however, that the pre-6.0 parser and the `-auditor` option will no longer be supported at the next major release of ObjectStore.

The new default front-end parser requires you to use the `OS_MARK_SCHEMA_TYPE` and `OS_MARK_SCHEMA_TYPESPEC` macros, as documented in the *ObjectStore C++ API Reference*, Chapter 4 ("System-Supplied Macros"), and in *Building ObjectStore C++ Applications*, Chapter 2 ("Working with Source

Files"). For a detailed description of the `ossg` utility, see "`ossg`: Generating Schemas" in *Managing ObjectStore*, Chapter 4 ("Utilities").

Note

If your makefile invokes `ossg` with the `-edgfe` option, you must edit the makefile to remove this option.

Improved Support for Failover

Release 6.1 provides improved support for failover, as follows:

- If you are using failover that is built into ObjectStore, you can configure failover so that the primary server and the secondary (or standby) server share the load during normal ObjectStore operations. In the event of failover, the secondary server assumes the full load.
- If you are using ObjectStore on the Sun Clusters 3.0 operating system (see Support for the Sun Clusters 3.0 on page 3), you can configure ObjectStore to use the support for failover provided by the operating system as an alternative to ObjectStore-managed failover.

For detailed information about failover, see *Managing ObjectStore*, Chapter 6 ("High Availability of Data").

ObjectStore's support for failover also allows you to perform rolling upgrades when installing a new release of ObjectStore. A rolling upgrade allows you to install a new release of ObjectStore without interrupting service to clients. For more information, see "`osconfig`: Configuring ObjectStore" in *Managing ObjectStore*, Chapter 4 ("Utilities").

Replication API

Release 6.1 introduces the following classes for managing database replication within an application:

- `os_replicator`
- `os_replicator_options`
- `os_replicator_statistic_info`

For more information about these classes, see `os_replicator` in *C++ API Reference*, Chapter 2 ("Class Library").

Defragmenting ObjectStore Databases

Release 6.1 provides support for defragmenting ObjectStore databases. This support includes:

- The `os_database::get_fragmentation()` function, which returns statistics on database fragmentation.
- The new `-f` option to the `ossize` utility. This option causes `ossize` to call the `os_database::get_fragmentation()` function.
- New server parameters that can be used to prevent fragmentation:
  - `Cluster Growth Policy`
  - `Database File Growth Policy`
  - `RAWFS Partition Growth Policy`
- New functions that can also be used to prevent fragmentation:
  - `os_cluster::set_size()`
  - `os_database::set_size()`
  - `os_database::set_size_in_sectors()`
- You can access these functions from the command-line with the `osdbcontrol` utility, using two new options: `-cluster_size` and `-size`

For information about defragmenting ObjectStore databases, see "Managing Database Fragmentation" in Chapter 1 ("Overview of Managing ObjectStore") of *Managing ObjectStore*.

Support for the Sun Clusters 3.0

Release 6.1 supports Sun Clusters 3.0. This support enables ObjectStore to use the failover support that is built into the Sun Clusters operating system. For more information, see Improved Support for Failover on page 2.

Architecture Sets for Release 6.1

Release 6.1 supports the following categories of architecture sets for use with the `ossg -arch` option when neutralizing schema:

- Standard architecture sets
- Versioned architecture sets
- User-defined architecture sets

In addition to the above architecture sets, Release 6.1 supports two other sets for use when neutralizing schema that includes virtual base classes; see Neutralizing Virtual Base Classes on page 4.

Note also that `ossg` has a new option, `-showsets`, which lists all architecture sets and their contents.

Standard Architecture Sets

The standard architecture sets meet the needs of most applications that require neutralization. These sets are:

- `all32`
- `all64`

Note that the `all` set is no longer supported. Support for this set has been removed because `ossg` no longer supports neutralization across all 32-bit and 64-bit platforms for applications that use the collections facility. If you use the `all` set in a makefile or any other scripts, you must remove it and substitute either `all32` or `all64`.

Versioned Architecture Sets

Starting with Release 6.1, you can use a versioned architecture set. A versioned architecture set contains only those platforms that are supported on a particular release of ObjectStore. For example, `all32_610` contains only those 32-bit platforms that are supported on Release 6.1. Unlike the contents of standard architecture sets, the contents of versioned architecture sets do not change from release to release.

For more information, see "Versioned Architecture Sets" in *Building ObjectStore C++ Applications*, Chapter 5 ("Building Applications for Multiple Platforms"). You can list all architecture sets and their contents by invoking `ossg` with the `-showsets` option, as described in "`ossg`: Generating Schemas" in *Managing ObjectStore*, Chapter 4 ("Utilities").

User-Defined Architecture Sets

Release 6.1 allows you to define your own architecture sets, which can be specified as arguments to `ossg`'s `-arch` option. To define an architecture set, use the `OS_USER_ARCH_SET` environment variable, as described in *Managing ObjectStore*, Chapter 3 ("Environment Variables").

Neutralizing Virtual Base Classes

If you are neutralizing schema against all 32-bit platforms, and the schema contains virtual base classes that use other virtual base classes, the `ossg` schema generator for Release 6.1 will prompt you to replace the virtual base classes with forced-order base classes. Replacing virtual base classes with forced-order classes ensures that the allocation order for the virtual base classes is the same across all 32-bit platforms.

Forced-order base classes are needed because of differences in the way different platforms allocate the virtual base classes. On linux3 (gcc3) platforms, virtual base classes are allocated in inheritance order. Currently, all other 32-bit platforms use post-traversal order. This difference results in a different layout order that requires neutralization.

If you are neutralizing against all 32-bit platforms *except* linux3, and your schema contains virtual base classes that use virtual base classes, you can specify the `all32vbtrav` architecture set to prevent the need for forced-order classes during neutralization. (The use of forced-order classes can increase code size.) For more information, see "Neutralizing the Allocation Order of Virtual Base Classes" in *Building ObjectStore C++ Applications*, Chapter 5 ("Building Applications for Multiple Platforms"). For more information about architecture sets, see Architecture Sets for Release 6.1 on page 3.

Changing the Windows Registration Location

Release 6.1 enables you to change the Windows registry location that is used by ObjectStore. Changing the registry location is especially useful when you have embedded ObjectStore in an application. By changing the registry location, you can prevent another application that also uses ObjectStore from overwriting information in the registry location that your application uses.

You can use the following to change the registry location:

- `os_authentication` class — For more information about this new class, see the description of the class in Chapter 2 ("Class Library") of the *C++ API Reference*.
- `osserver -r` — For more information about the new `-r` option to `osserver`, see "osserver: Starting the Server" in Chapter 4 ("Utilities") of *Managing ObjectStore*. Note that the `oscmgr6` utility for starting the cache manager now has the same new `-r` option.
- `OS_REMOTE_AUTH_REGISTRY_LOCATION` — For more information about this new environment variable, see "osserver: Starting the Server" in Chapter 3 ("Environment Variables") of *Managing ObjectStore*.

For information about changing the registry location on NT machines, see "Setting the Registry Location for ObjectStore (Windows Only)" in *Managing ObjectStore*, Chapter 8.

Preventing Excessive Page Faulting

Release 6.1 provides the following functions of the `objectstore` class that enable you to prevent excessive page faults by disabling address markers:

- `objectstore::get_asmarkers_useless()`
- `objectstore::set_asmarkers_useless()`

You can also use a new environment variable, `OS_ASMARKERS_USELESS`, to disable address markers. For information about the environment variable, see the description in Chapter 3 ("Environment Variables") of *Managing ObjectStore*. The functions are described in Chapter 2 ("Class Library") of the *C++ API Reference*.

Changes to Dump and Load

Release 6.1 includes enhanced versions of the ObjectStore `osdump` and `osload` utilities. Changes to these utilities include:

- Support for multiple processors in both `osdump` and `osload`. The option to specify the use of multiprocessors is `-pr`.
- Support for the ability to restart the `osload` process.
- The mechanism for generating the source code for loader applications is moved from `osdump` to `osload`. This option to specify this operation remains `-emit`.
- Elimination of the `-ds` option to dump the database schema. The database schema is always dumped in Release 6.1.

For more information, see `osdump` and `osload` in Chapter 4 ("Utilities") of *Managing ObjectStore*.

Using Dump and Load to Migrate Databases

If you plan to use `osdump` and `osload` to migrate databases from Release 5.1 or release 6.0, you should always check the ObjectStore Technical Support Web site (`www.objectstore.net/support`) for the most recent version of each of these utilities.

The dump and load subsystem has undergone considerable revision since the last release. Data files dumped with prior releases of `osdump` are not compatible with current and future releases of `osload`. The databases from which those files were generated will need to be re-dumped with an updated version of `osdump`.

Updated versions of `osdump` for ObjectStore 5.1 and ObjectStore 6.0 can be obtained by contacting ObjectStore Technical Support. If you need to run `osdump` or `osload` with more than one process under any version of Microsoft Windows, you need to obtain updated versions from support as well. If you need to upgrade an OSJI database on any platform you must obtain updated libraries as well.

Applications built or linked with `libosdump` or `libosload` will at the very least need to be recompiled and relinked to take advantage of recent bug fixes. The following APIs have changed as well and may require code changes for some applications:

- `os_Database_table` has changed to allocate at the cluster level rather than the segment so any function calls which used to take an `os_Segment` as an argument will now take an `os_Cluster`.
- `os_Database_table::insert()` has been replaced with explicitly named function calls. This eliminates the confusion of 12 different overloadings of the `insert()` function.
- `os_Loader::load()` has been changed to `os_Loader::start_load()`.

New osload Features

New features of `osload` in 6.1 include

- resumption of work in the event of failure
- an easier process to generate a schema specific osload
- fewer required command line arguments.

Resumption of `osload`

In the event of failure during load, do not remove any temporary databases and simply re-start `osload`. The `osload` application will look for a current work database and attempt to resume based on the state of the found work database. In general this is only useful if the failure is due to hardware or user failures , such as out of disk space, network failure, or user interruptions.

Easier schema generation

The updated process to generate a schema specific `osload`:

1. Generate a 6.1 application schema; for example, `my_new_schema.adb`
2. Emit the `osload` specializations required for that schema; for example:

```
osload -emit my_new_schema.adb
```

3 Compile the new `osload` with the files generated in step 2. You will need to update the makefile with your application specific libraries and headers.

Simplified `osload` usage

The usage of `osload` has changed and it no longer requires you to input the dumped file names. By default `osload` will look for a file named `db_table.dmp` in the current directory and, using that master file, it builds the work file list. If the files are located in a directory other than the current directory the `-dir` switch must be used.

Performance Improvements

New features of `osdump` and `osload` in ObjectStore 6.1 and post-ObjectStore 5.1.5 include performance and scalability improvements. General performance improvements require no additional steps, while scalablity improvemnts are acheived by running `osdump` or `osload` with more than one process. To do so use the `-pr` switch and provide the number of process to do the work (for example. `-pr 2`).

Note on Multi-process osdump and osload

There is a fair amount of overhead necessary to manage the child processes in the 6.1 release so it may at times be slower than running `osload` with a single process. (It is expected that this overhead will be reduced in the future). The dump and load work is partitioned among child processes by the segments in a supplied database. In general, if the segments to be dumped or loaded are large (i.e., greater than 100 MB) and you have more than one in a given database, you should use multi-process `osdump` or `osload`. On the other hand, if you have a database with 2000 segments of approximately 5 MB each or a database with one segment approximately 2GB in size, you should use osdump or osload with a single process.

New Macro for Functions Used in Queries

Release 6.1 provides a new macro, `OS_MARK_QUERY_FUNCTION_WITH_NAMESPACE()`, for use in schema source files to include a query function in the schema. Use this macro instead of the `OS_MARK_QUERY_FUNCTION()` macro when the function is a member of a class that is declared in a namespace. For more information about the `OS_MARK_QUERY_FUNCTION_WITH_NAMESPACE()` macro, see the *C++ Collections Guide and Reference*, Chapter 9.

Optimizing Collections

Previous to Release 6.1, you could optimize the creation of transient sets, lists, and cursors by manually adding an optimization flag to the constructor. The flag would instruct ObjectStore to use hard pointers instead of soft pointers for the constructed type.

Release 6.1 introduces the following changes for optimizing transient sets, lists, and cursors:

- The `os_cursor::optimized` flag is no longer needed to construct an optimized transient cursor. By default, all transient cursors use hard pointers.
- Two new defines have been added for the release: `_OS_COLL_LIST_OPTIMIZE` and `_OS_COLL_SET_OPTIMIZE`. By setting these defines in your application or on the compile line, users can globally change their applications to use transient lists and sets that are based on hard pointers, without having to specify the optimization flags in the constructors.

For more information about these optimizations, see "Compiling for Collections Optimization" in *C++ Collections Guide and Reference*, Chapter 2.

Compiling Single-Threaded Applications on Solaris

When using the Solaris compiler to compile single-threaded applications, it is no longer necessary to use the `-mt` option. You must use this option when compiling a Release 6.1 application only if your application uses multiple threads. Refer to the Solaris documentation for more detailed information about the `-mt` option.

Note that, if you want a Release 6.1 application to use a Release 6.0 cache manager, you must compile the application with the `-mt` option, even if it is single-threaded. In this situation, however, the performance of the client/cache interface will be the same as for a 6.0 application that uses a 6.0 cache manager — not as fast as for a 6.1 application that uses a new 6.1 cache manager. For best performance and to avoid the overhead of linking with the ObjectStore thread-safe library (`libosth`), 6.1 single-threaded applications should use the 6.1 cache manager.

objectstore::export() Renamed

The `objectstore::export()` function has been renamed `objectstore::export_object()` to reflect its purpose more clearly — to export persistent, top-level objects. For more information about the function,

see its description in Chapter 2 (“Class Library”) of the *C++ API Reference*. If you wish to continue using the old name, you must define `OS_EXPORT_API` at the top of your source file, before the ObjectStore header files are included.

Changes to the ObjectStore Java Interface (OSJI)

Release 6.1 of the ObjectStore Java Interface (OSJI) now supports both JDK 1.3 and 1.4. Support for JDK 1.2 and earlier is no longer maintained.

Changes to Javlin/JMTL

In Release 6.1, Javlin and the Java Middle Tier Library (JMTL) have been incorporated into the ObjectStore product and use the same release numbering.

com.odi.jmtl.env.JVMEnviroment Class Renamed

The `com.odi.jmtl.env.JVMEnviroment` class has been renamed to `com.odi.env.JVMEnviroment`. Java applications that use the `JVMEnvironment` class need to be modified and recompiled.

com.odi.jmtl.env.JVMEnviroment.initialize() Method Renamed

The `com.odi.jmtl.env.JVMEnviroment.initialize()` method is replaced by `com.odi.env.JVMEnviroment.deploy()`. Any Java classes using this method need to be modified and recompiled.

New Javlin/JMTL Deployment Descriptor Format

With Release 6.1, Javlin/JTML has a new deployment descriptor format. The new deployment descriptor format requires the following XML modifications:

Release 1.2 Element	*Release 6.1 Element*
`CachePoolManager`	`cache-pool-manager`
`CachePools`	`cache-pools`
`CachePool`	`cache-pool`
`CacheStorage`	`cache-storage`
`Databases`	`databases`
`DatabaseDescriptor`	`database-descriptor`
`Roots`	`roots`
`RootDescriptor`	`root-descriptor`

Release 1.2 Element	*Release 6.1 Element*
`RootObjectDescriptor`	`root-object-descriptor`
`Component`	`component-descriptor`
`Method`	`method`
`MethodDescriptor`	`method-descriptor`
`ExtentDescriptor`	unsupported
`FinderDescriptor`	unsupported

The new deployment descriptor format is defined by the DTD file `jmtl-dd.dtd` located in `jmtl.jar`. To use the new format, change the `DOCTYPE` entry from:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/xml/EnvConfig.dtd">
```

to:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/jmtl-dd.dtd">
```

Release 6.1 includes a DTD file `jmtl-dd-compat.dtd` that you can use to make existing Javlin/JMTL 1.2 XML deployment descriptor files compatible with Release 6.1. To use this file, change the `DOCTYPE` entry from:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/xml/EnvConfig.dtd">
```

to:

```
<!DOCTYPE JVMEnvironment SYSTEM "JMTL:com/odi/jmtl/jmtl-dd-compat.dtd">
```

New Cache Pool Attributes

Release 6.1 includes new cache pool attributes that influence cache performance. The new attributes are:

- `CommitIfIdle`
- `CommitIfIdleMvcc`
- `GroupOpenInterval`
- `GroupOpenIntervalMvcc`
- `LockTimeout`
- `MaxConcurrentTransactions`

For more information on these attributes, see “Declarative Configuration” in the *Javlin User Guide*.

Integration with WebLogic Server 7.0

Release 6.1 includes a new application server specific jar file that provides level one integration with BEA WebLogic Server 7.0.

Updated Examples Using Ant Build Files

The Javlin/JMTL examples have been updated to use Apache Ant to simplify configuration, building, and deployment. For additional information on using Ant, see the `README.html` file in the examples directory.

Changes to the Documentation

The following changes have been made to the documentation for Release 6.1:

- The ObjectStore online documentation has been reorganized into two bookshelves:
  - C++ bookshelf
  - Java bookshelf
- The online documentation is delivered through WebHelp, which provides full-text search capability.
- All of the documentation has been consolidated into the `doc` directory. This directory is directly under the installation directory for ObjectStore and contains the documentation for all installable components of this release, regardless of whether or not you decide to install them all.
- The following features of ObjectStore were previously released but never documented until this release:
  - `os_schema_evolution::set_explanation_level()`: See the description in Chapter 2 (“Class Library”) of the *C++ API Reference*.
  - `os_schema_evolution::set_resolve_ambiguous_void_pointers()`: See the description in Chapter 2 (“Class Library”) of the *C++ API Reference*.
  - `osconfig`: See the description in Chapter 4 (“Utilities”) of *Managing ObjectStore*.
- The *Javlin User Guide* contains a new chapter titled “Chapter 7: Using the Javlin Console”

Summary of Changes

The following sections summarize all user-visible changes to ObjectStore for Release 6.1, including changes that have been more fully described in the preceding sections. The information in these sections is provided as a migration aid for users who might want to know at a glance how Release 6.1 will impact their applications and scripts. Each section also includes

references to the relevant parts of the ObjectStore documentation for more detailed information about the changes.

New C++ Classes and Functions

The following classes and functions have been added to the ObjectStore API for Release 6.1:

- `objectstore::get_asmarkers_useless()`
- `objectstore::set_asmarkers_useless()`
- `os_authentication`
- `os_cluster::set_size()`
- `os_database::get_fragmentation()`
- `os_database::set_size()`
- `os_database::set_size_in_sectors()`
- `os_dbutil::install_backrest_control_c_handler()`
- `os_dbutil::start_backrest_logging()`
- `os_dbutil::stop_backrest_logging()`
- `os_dbutil::svr_machine()`
- `os_replicator`
- `os_replicator_options`
- `os_replicator_statistic_info`
- `os_schema_evolution::set_explanation_level()`
- `os_schema_evolution::set_resolve_ambiguous_void_pointers()`

Some of these functions are described in other sections of this document. For detailed information about all of the new classes and functions, see the descriptions in Chapter 2 (“Class Library”) of the *C++ API Reference*.

New Environment Variables

The following environment variables are new for Release 6.1:

- OS_16K_PAGE
- OS_32K_PAGE
- OS_64K_PAGE
- OS_8K_PAGE
- OS_ASMARKERS_USELESS
- OS_CORE_DIR
- OS_NETWORK_SERVICE
- OS_PREALLOCATE_CACHE_FILES
- OS_REMOTE_AUTH_REGISTRY_LOCATION
- OS_USER_ARCH_SET

Some of these environment variables are described in other sections of this document. For detailed information about all of the new environment variables, see their descriptions in Chapter 3 (“Environment Variables”) of *Managing ObjectStore*.

New Server Parameters

The following server parameters are new for Release 6.1:

- Cluster Growth Policy
- Database File Growth Policy
- Failover Heartbeat File
- Failover Script
- Identical Pathnames on Failover Server
- RAWFS Partition Growth Policy
- RPC Timeout

Some of these server parameters are described in other sections of this document. For detailed information about all of the new server parameters, see their descriptions in Chapter 2 (“Server Parameters”) of *Managing ObjectStore*.

New Options for ObjectStore Utilities

New options have been added to the following ObjectStore utilities:

- osarchiv -P -T

- `osbackup -P -r -T`
- `oscopy rawfs_dir`
- `osdbcontrol -cluster_size -size`
- `osdump -dir -pr -v`
- `osload -dir -pr -v`
- `osrecovr -s -T`
- `osreplic -c -C -P -R -s -T`
- `osrestore -s -T`
- `osserver -con -hostname -M -server_name -upgradeRAWFS`
- `ossg -auditor -showsets -ignore_vbo`

  Note that the `-auditor` option is provided as a migration tool and is not described in *Managing ObjectStore*. For information about this option, see ossg's Default Front-End Parser on page 1.

- `ossize -f`
- `ossvrstat -databases`
- `osverifydb -all -tag`

For information about the new options refer to the descriptions of the utilities in Chapter 4 ("Utilities") of *Managing ObjectStore*.

Unsupported C++ Functions

The `objectstore::export()` function is not supported and has been replaced by the `objectstore::export_object()` function. For more information, see objectstore::export() Renamed on page 9.

Unsupported Java Methods

The `JVMEnvironment.initialize()` method is not supported and has been replaced by the `JVMEnvironment.deploy()` method. For more information, see com.odi.jmtl.env.JVMEnviroment.initialize() Method Renamed on page 10.

Unsupported Options for ObjectStore Utilities

The following options to the ossg utility are no longer supported:

- The `-edgfe` option; for more information, see ossg's Default Front-End Parser on page 1.

- The `-store_function_parameters` (`-sfp`) option. Specifying the `-store_member_functions` (`-smf`) option provides the same functionality; see the description of the `ossg` utility in Chapter 4 ("Utilities") of *Managing ObjectStore*.

Restrictions, Limitations, and Known Problems

The following sections describe restrictions, limitations, and known problems when using Release 6.1. Where possible, they also describe workarounds for the problems.

For more up-to-date information about Release 6.1, see the Support Matrix on the Technical Support web site (`www.objectstore.net/support/matrix`).

Incompatibilities Between Visual C++ 6 and 7

Because of changes in Microsoft's Visual C++ compiler from version 6 (vc6) to version 7 (vc7), users who upgrade their applications from vc6 to vc7, or who wish to use both versions, may see schema incompatibilities that affect schema generation. The following sections discuss these incompatibilities and their workarounds.

Differing Support for #pragma pack(pop,N)

Support for `#pragma pack(pop,`*`N`*`)`, where *`N`* is some alignment, differs between vc6 and vc7. In particular, the following set of pragmas results in different alignments between vc6 and vc7:

```
#pragma pack(push,l1,2)
#pragma pack(push,l2,4)
#pragma pack(pop,1)
```

On vc6, these pragmas result in a two-byte alignment; whereas on vc7 they result in a one-byte alignment. The workaround for this incompatibility is to replace the last pragma with either of two sets of pragmas. The first set ensures one-byte alignment that is compatible with vc7:

```
#pragma pack(pop)
#pragma pack(1)
```

The next set ensures two-byte alignment that is compatible with vc6:

```
#pragma pack(pop)
#pragma pack(2)
```

Differing Support for Integral Extension Types

On vc6, the integral extension types are distinct types. On vc7, they are treated as typedefs for regular C++ types. Thus, the following code behaves differently on vc6 and vc7:

```
template<class T> class printer {
public:
  static void print(const char* source_name) {
    const type_info& ti = typeid(T);
    printf("in source %s, actual %s\n",ti.name());
  }
};
printer<__int16>::print("__int16");
printer<unsigned __int32>::print("unsigned __int32");
```

On vc6, the output is:

```
in source __int16, actual __int16
in source unsigned __int32, actual unsigned __int32
```

On vc7, the output is:

```
in source __int16, actual short
in source unsigned __int32, actual unsigned int
```

Furthermore, the symbol for `print` (or for the virtual function table, if there was one) is different. On vc6, the template argument for `__int16` would be `_F`; on vc7, it would be `F`.

If you wish to share code or databases between vc6 and vc7 applications, you cannot use integral extension types in any context that appears in the schema, including template instantiations and virtual function table pointers. If you are not sharing code between vc6 and vc7 applications, you can continue to use the integral types and the “mangling” will follow the rules for the compiler version you are using.

Red Hat Linux 8 Address Space Limitation

On Red Hat Linux 8, the default setting for the `vm.max_map_count` kernel parameter (65536) limits the address space to 256 MB. This can create a potential address space limitation. You can change the value of this parameter with the following command line:

```
sysctl -w vm.max_map_count= value
```

To display all kernel parameters, use the following command line:

```
sysctl -a
```

You can also work around the address space limitation by setting the ObjectStore environment variable `OS_8K_PAGE` or `OS_16K_PAGE`. Setting either of these variables has the effect of increasing the page size read by ObjectStore to 8 KB or 16 KB. In some cases, increasing the page size might result in additional lock conflicts because ObjectStore creates locks on a per-page basis. A page size of 8 KB or 16 KB can have greater chances of lock conflicts than a page size of 4 KB.

Multi-process Dump and Load

Release 6.1 does not support the use of multi-process dump and load on Windows platforms.

Address-Space Release Facility

The ObjectStore address-space release facility is not thread safe. As a result, there is the risk that one thread will release address space that is being used by another, leading to unpredictable behavior and potential database corruption. To protect against this risk, this release of CMTL disables the address-space release in software with which it is linked. It is possible that this protective measure can impose address-space limitations that might prevent a 32-bit CMTL application from being successfully deployed. A more general solution to the problem will be implemented in a future release of CMTL.

Generating Schema for Empty Abstract Classes

The schema generator (`ossg`) cannot generate schema for code that contains empty abstract classes used as virtual base classes on Solaris platforms. As a workaround, `ossg` will emit instructions to add a padding member.

Least-Space Allocation Strategy

You should not use the least-space allocation strategy until contacting ObjectStore Technical Support for a patch. Note that the least-space allocation strategy is not the default. The default is the least-wait allocation strategy, which you can continue to use.

For information about allocation strategies and how to set them see `objectstore::set_allocation_strategy()` in Chapter 2 (“Class Library”) of the *C++ API Reference*.

Vector Header Restrictions

Under the following conditions, corrupted vector headers can result:

- A vector was created on Solaris SPARC 2.8 (64-bit) without using the normal form of the ObjectStore overloaded `::operator new`. The normal form is the `::operator new` followed by constructor syntax, as in the following example:

  ```
  void *p = new(db, foo::get_os_typespec(), N) foo[N];
  ```

  When this normal form is used, there is no problem. Problems occur with forms such as:

  ```
  void *p = ::operator new(
           sizeof(foo)*N, db, foo::get_os_typespec(), N);
  ```

- A vector was hetero-relocated from a different platform to Solaris SPARC 2.8 (64-bit) or Linux GPP, and the page containing the vector was committed.

To detect and correct vector header problems, use the `osfixvh` utility.

Allocator Framework Restrictions

The allocator framework for Release 6.1 does not support the following:

- Sessions
- Threads
- Multiple context matching, as described in "Rule Matching" of the *Advanced C++ API User Guide*, Chapter 10 ("Allocator Framework").

Support for these features will be provided at a future release of ObjectStore.

CMTL Restrictions

The following sections describe restrictions when using the Release 6.1 version of CMTL.

Configuring CMTL from XML on Linux Platforms

On Linux platforms, if your application configures CMTL from an XML-based configuration stream, you must specify the `ios::in` flag when creating the `ifstream` object that is passed as an argument to `os_cache_pool_manager_configuration::create from_xml_stream()`.

On Linux platforms, `ifstream` is not opened for reading by default. The workaround is to explicitly pass an argument to the `ifstream` constructor telling it to open the stream for reading, as in the following example:

```
ifstream xml_file(config_info, ios::in);
```

The `ios::in` flag tells the constructor to open the stream for reading. Note that adding this flag to the constructor is required only on Linux platforms.

Setting the commit_if_idle attribute

The CMTL cache pool attribute `commit_if_idle` value default is `true` for read-only caches (`true` is also the value for update caches). Setting the `commit_if_idle` attribute value to `false` will result in a shutdown timing problem in the CMTL virtual transaction manager thread. If you have explicitly set the `commit_if_idle` value to false in your cache pool configuration, you need to modify it to avoid this timing issue.

DDML Restrictions

The following restrictions apply to the Release 6.1 version of DDML:

- DDML is not supported on Solaris 64-bit (sol64) platforms.
- The following DDML warning message can be safely ignored:

```
Warning:  com.odi.osdm.JosdmCPlusPlus.charPmap is a static
field of type com.odi.util.OSSmallMap which might refer to a
persistent object.  If this field does refer to a persistent
object it must be user maintained.
```

Java Components of ObjectStore

The following sections describe the restrictions concerning the Java components of ObjectStore — for example, OSJI and Javlin.

Upgrading Pre-6.0 OSJI Databases to Release 6.1

This release does not support the use of the dump/load facility for the purpose of upgrading a pre-6.0 OSJI database to Release 6.1. If you have a pre-6.0 OSJI database that you want to upgrade, you must contact ObjectStore Technical Support.

Not Supported on 64-Bit Platforms

The Java components of ObjectStore are not supported on 64-bit platforms for this release.

Running Java Browser on UNIX

Running the Java browser on UNIX platforms from Exceed clients have the following problems:

- The Message Box is Empty, because the Windows Manager does not resize the Message Box correctly. You have to resize the Message Box so the content will appear.
- Menu position is incorrect. You need to resize the main window so the menu position is correct.

Use of JDK 1.2 and 1.4 on Solaris

To ensure compatibility between ObjectStore and JDK 1.3 or JDK 1.4 on Solaris, you must set the `LD_PRELOAD` environment variable as follows:

```
setenv LD_PRELOAD libosopdel.so
```

Level One Integration

Level one integration (JTA transaction integration) is supported for BEA WebLogic Server 6.1 and BEA WebLogic Server 7.0 only.

Terminated Sessions Do Not Release All Resources

Currently, a terminated session does not release all of its resources until each thread in a session explicitly calls the `Session.leave()`, `Session.terminate()`,`Objectstore.shutdown()`,`Session.create()` or `Session.join()` method.

Terminating a session continues to make all threads leave the session for all other purposes, except for releasing the session's resources.

The `Session.leave()` method has been modified so it can be called by threads, even if they are not joined to a session. Previously, an exception would be thrown.

Applications must ensure that all threads explicitly leave sessions that have been terminated, so that session resources are freed. If this is not done, resources allocated to sessions are not released, resulting in a possible `com.odi.AddressSpaceFullException` being thrown when trying to create a new session.

Threads Are Not Being Automatically Joined to

Nonglobal Sessions

Threads that do not belong to a session are not automatically joined to a nonglobal session when they should be. Until this problem is resolved, applications cannot rely on session absorption to make a thread that accesses persistent objects join the appropriate session. As a work around, applications can explicitly join each thread to a session by calling `Session.join()` or using a global session.

More specifically, API methods whose only session-implying arguments are persistent objects require that the calling thread already belong to the same session as the persistent objects. This applies to nonglobal sessions. This restriction will be lifted in a future release. The methods affected by this restriction include the following:

- `ObjectStore.deepFetch()`
- `ObjectStore.destroy()`
- `ObjectStore.dirty()`
- `ObjectStore.evict()`
- `ObjectStore.export()`
- `ObjectStore.fetch()`
- `ObjectStore.isDestroyed()`
- `ObjectStore.isExported()`
- `ObjectStore.migrate()`
- `Persistent.deepFetch()`
- `Persistent.destroy()`
- `Persistent.dirty()`
- `Persistent.evict()`
- `Persistent.fetch()`
- `Persistent.isDestroyed()`

Schema Write-Lock Conflicts Might Occur Immediately Following Schema Installation

Users running ObjectStore applications concurrently against a database might encounter schema segment write-lock conflicts during a transaction that immediately follows a transaction in which the schema was installed.

To work around this problem, run a small dummy transaction immediately after the transaction that installed the schema. The dummy transaction must

use the database, such as looking up a database root. Using the database validates the schema and prevents potential future write lock conflicts.

Use of ossevol Utility

Do not use the `ossevol` utility on databases created with the Java interface to ObjectStore (OSJI). Also, do not use the C++ API for this utility on OSJI databases.

You can use the OSJI `Database.evolveSchema()` method to evolve the schema in an OSJI database.

Use of osgc and oscompact Utilities

You should not run the `osgc` or `oscompact` utilities against databases that are currently opened with applications that retain references to non-exported objects. To prevent possible corruption you should close the databases before running either utility.

Hosted Pathname Syntax Might Require Setting of Environment Variable

If the directory specified in `OS_LIBDIR` uses the hosted pathname syntax (host:/ dir) and the pathname syntax for that directory has the opposite style of slash from the one for local pathnames on the client (that is, Windows and UNIX), you must set the `OS_META_SCHEMA_DB` environment variable to the pathname of the metaschema database. That database is named `metaschm.db`, and its default location is in the lib subdirectory of `OS_ROOTDIR`.

Transient Segmentation Violation Errors

On UNIX platforms there are some known interaction problems between the Java VM and OSJI. If you encounter any *transient segmentation violations* errors, choose a heap size and set both the initial and maximum heap size to that value on the command line. For example, if you choose a 64 MB heap size, specify both `-Xms64m` and `-Xmx64m` as arguments to the java command.

Applets

ObjectStore is a Java application that uses C++ native methods. Consequently, you cannot use ObjectStore in an applet other than through the Sun JDK Appletviewer application.

Troubleshooting Problems - It Might Be the JIT Compiler

Just In Time (JIT) compilers often make the difference between acceptable and unacceptable performance. However, if you are having undiagnosed trouble when using a JIT compiler, toggling the JIT compiler on and off might pinpoint the JIT compiler as the source of the problem.

Object Design has tested the JIT compilers available with ObjectStore's supported and maintained platforms. Some JIT compilers tested by Object Design have exhibited problems when running tests of ObjectStore. These problems often appear when NullPointerException is signaled unexpectedly, although other incorrect behavior has also been seen. Object Design has fixed or worked around the ObjectStore problems that have been encountered. However, you might still encounter new problems in ObjectStore or in your application when you use a JIT compiler.

Object Design's Technical Support might be able to assist you with JIT compiler problems by recommending ways to work around the troublesome behavior, but you might need to contact the compiler's vendor about the issue.

The following sections describe how to toggle off the JIT compiler to diagnose troublesome behavior.

Disabling JIT on Solaris

To disable the JDK JIT on Solaris, do one of the following:

- Set the `JAVA_COMPILER` environment variable to `NONE`:

  ```
  setenv JAVA_COMPILER NONE
  ```

- Specify `NONE` as the value of the `java.compiler` system property:

  ```
  java -Djava.compiler=NONE MyClass
  ```

Disabling Sun JDK JIT on Windows

To disable the Sun JDK JIT on Windows, do one of the following:

- Unset the `JAVA_COMPILER` environment variable:

  ```
  set JAVA_COMPILER=
  ```

- Specify `NONE` as the value of the `java.compiler` system property:

  ```
  java -Djava.compiler=NONE MyClass
  ```

Peer Generator Incorrectly Generates Code for Certain

Abstract Classes

Under certain circumstances, the peer generator tool (`osjcgen`) fails to recognize that a class is abstract, so it generates code that attempts to instantiate the class. The generated C++ code does not compile.

A workaround is to suppress the generation of the methods that contain compilation errors. To do this, specify the `-suppress` option with the name of a problem method when you run the peer generator tool. This prevents the problem methods from being generated.

For example, suppose the C++ `person` class has a Java peer class generated into the `com.people package` and there are three methods that are not compiling correctly. Run `osjcgen` and specify the `-suppress` option for each problem method.

```
-suppress com.people.person.person \
-suppress com.people.personU.makeArray \
-suppress com.people.personU.set

The problem occurs when a C++ class inherits a pure virtual
function from a base class. For example:


/* class A is abstract */
class A {
  public:
    virtual void f() = 0;
}

/* class B is abstract, but osjcgen treats it as nonabstract */
class B : public A {
  public:
    virtual void f() = 0;
}

/* class C is abstract - the redeclaration of the pure virtual
  causes osjcgen to handle this correctly. */
class C : public B {
  public:
    virtual void f() = 0;
}

/* class D is nonabstract */
class D : public C {
  public:
    virtual void f();
}
```

Postprocessor Options Required for ObjectStore Peer

Collections with Indexes

If you use indexes with ObjectStore peer collections, you must specify the `-nothisopt` and `-noarrayopt` options when you run the postprocessor on your classes. Alternatively, you can specify the `-noinitializeropt` option in place of the two options.

This ensures that the postprocessor does not apply certain optimizations, which might cause your code to work incorrectly for evict operations performed on ObjectStore collections. These evict operations can happen during execution of the following methods:

- `addIndex()`, `query()`, `queryPick()`, and `exists()` on any collection
- `insert()`, `replaceAt()`, `insertFirst()`, `insertLast()`, `insertBefore()`, and `insertAfter()` on collections with indexes that have `SIGNAL_DUPLICATES` behavior

Solaris: Accessing Multithreaded C++ Applications

When OSJI applications access C++ libraries on Solaris, those libraries must be linked with `libosthr` to work with the native threads version of the Java virtual machine.

The C++ interface to ObjectStore (OSC++) provides two thread libraries:

- `libosthr` ensures that calls to ObjectStore from multiple C++ threads are safe.
- `libosths` does not provide this protection.

To be used by an OSJI application running with the native threads version of the Java virtual machine, multithreaded OSC++ libraries must use `libosthr`.

Using the native threads version of the Java virtual machine, you can link multithreaded OSC++ libraries with `libosthr` and depend on the locking primitives provided by OSC++. When you do this, the link line must include `-losthr` and either `-mt` or `-lthread`.

Destroying Java Peer Objects

There is a bug that prevents ObjectStore from leaving a tombstone when you destroy a Java peer object. This bug will be fixed in a future release.

For now, you must be careful that you do not destroy a Java peer object that is still referred to by another object and then try to use that reference. While doing so is always a mistake, in the current product there is no tombstone to flag the mistake for Java peer objects.

Using the Documentation with Netscape Browsers

The ObjectStore documentation set is compatible with Netscape Release 4.7x and Release 6.1x. The documentation set cannot be properly displayed with Netscape Release 6.0 on any platform. If you only have access to Netscape Release 6.0, you should use the PDF versions of the documentation set.

Platform and Release Compatibility

This section lists this platforms and compilers supported by this release of ObjectStore.

The Support Matrix on the Technical Support web site (`www.objectstore.net/support/matrix`) contains an up-to-date list of all supported and maintained platforms. Please refer to the Support Matrix if you are in any doubt whether your compiler or operating system are supported. If your compiler is not supported, you cannot use this release of ObjectStore.

Compiler Compatibility

If you are using a supported compiler and are a first-time user of ObjectStore, installing ObjectStore is a straightforward process. Likewise, if you are upgrading from Release 6.0 and are using a supported compiler that is the the same as the compiler you used for the previous release, the process of upgrading to Release 6.1 is straightforward.

If you are upgrading from a pre-6.0 release or your compiler has changed since the previous release, please refer to the *ObjectStore Migration Guide* (`www.objectstore.net/documentation/migration`) that is available on the Technical Support web site. The *ObjectStore Migration Guide* will provide detailed instructions about upgrading to Release 6.1.

Platform Configuration: Solaris 32-bit (SOL2C5)

You can build and run 32-bit or 64-bit applications on 64-bit hardware, but you cannot build or run 64-bit applications on 32-bit hardware

Supported Operating Systems	Solaris 8 Solaris 9

Supported Clusters	Sun Cluster 3.0 5/02 for Solaris 8
Supported C++ Compilers	Sun ONE Studio 7 (C++ 5.4)
Supported Java Compilers	Sun Java 2 SDK 1.3 or 1.4
Required Patches	Solaris 8 systems require Sun Patch 108434-*xx* for Sun ONE Studio 7 compiler use. Solaris 9 systems require Sun Patch 111711-*xx* for Sun ONE Studio 7 compiler use.
Recommended Patches	Sun Patch 108528-*xx* is recommended but not required for all Solaris 8 systems running ObjectStore for its resolution of BugID 449415. This patch is included in the current Solaris 8 recommended patch cluster.

Note

The string *xx* refers to the latest available revision of the patch from Sun.

If you are using ObjectStore's built-in failover or failover as provided by the Sun Clusters 3.0 operating system, you may be able upgrade to Release 6.1 without having to take your system out of service by performing a *rolling upgrade*. For more information, see *ObjectStore Installation for UNIX*.

Platform Configuration: Solaris 64-bit (SOL64)

You can build and run 32-bit or 64-bit applications on 64-bit hardware, but you cannot build or run 64-bit applications on 32-bit hardware

Supported Operating Systems	Solaris 8 Solaris 9
Supported Clusters	Sun Cluster 3.0 5/02 for Solaris 8
Supported C++ Compilers	Sun ONE Studio 7 (C++ 5.4)
Supported Java Compilers	Not supported yet

Required Patches	Solaris 8 systems require Sun Patches 108434-xx and 108435-xx for C++ compiler use. Solaris 9 systems require Sun Patch 111711-xx and 111712-xx for Sun ONE Studio 7 compiler use.
Recommended Patches	Sun Patch 108528-*xx* is recommended but not required for all Solaris 8 systems running ObjectStore for its resolution of BugID 449415. This patch is included in the current Solaris 8 recommended patch cluster.
Unsupported Components	OSJI Javlin

Note

The string *xx* refers to the latest available revision of the patch from Sun.

If you are using ObjectStore's built-in failover or failover as provided by the Sun Clusters 3.0 operating system, you may be able upgrade to Release 6.1 without having to take your system out of service by performing a *rolling upgrade*. For more information, see *ObjectStore Installation for UNIX*.

Platform Configuration: Windows 32-bit VC6 (WINVC6)

Supported Operating Systems	Windows NT 4.0 SP6a Windows 2000 Windows XP
Supported C++ Compilers	Microsoft Visual C++ 6.0 SP5
Supported Java Compilers	Sun Java 2 SDK 1.3 or 1.4

Platform Configuration: Windows 32 bit VC7 (WINVC7)

Supported Operating Systems	Windows 2000 Windows XP
Supported C++ Compilers	Microsoft Visual Studio .NET
Supported Java Compilers	Sun Java 2 SDK 1.3 or 1.4

Platform Configuration: Linux (LINX3)

Supported Operating Systems	Red Hat 8.0 with kernel 2.4.18 and glibc 2.293
Supported C++ Compilers	gcc 3.2
Supported Java Compilers	Not supported yet
Unsupported Components	OSJI Javlin DDML

Index

E

F

G

J

L

M

N

O

P

Q

R

S

T

U

V

W